Windowed aggregations over successively increasing timed windows

Status: *Draft*

```
KTable<Windowed<Key>, Value> oneMinuteWindowed = // where Key and Value stand for your actual key and value
types

        yourKStream

        .groupByKey()

        .reduce(/*your adder*/, TimeWindows.of(60*1000, 60*1000), "store1m");
                //where your adder can be as simple as (val, agg) -> agg + val
                //for primitive types or as complex as you need

KTable<Windowed<Key>, Value> fiveMinuteWindowed =

        oneMinuteWindowed
        .groupBy( (windowedKey, value) ->
                new KeyValue<>(
                        new Windowed<>(
                                windowedKey.key(),
                                new Window<>(
                                        windowedKey.window().start() /1000/60/5 *1000*60*5,
                                        windowedKey.window().start() /1000/60/5 *1000*60*5 + 1000*60*5
                            // the above rounds time down to a timestamp divisible by 5 minutes
                                )
                        ),
                        value
                ),
                /* your key serde */,
                /* your value serde */
        )
    .reduce(/*your adder*/, /*your subtractor*/, "store5m");
        // where your subtractor can be as simple as (val, agg) -> agg - val for primitive types
        // or as complex as you need,
        // just make sure you get the parameter order right, subtraction is not commutative!

KTable<Windowed<Key>, Value> fifteenMinuteWindowed =

        fiveMinuteWindowed

    .groupBy( (windowedKey, value) ->
                new KeyValue<>(
                        new Windowed<>(
                                windowedKey.key(),
                                new Window<>(
                                        windowedKey.window().start() /1000/60/15 *1000*60*15,
                                        windowedKey.window().start() /1000/60/15 *1000*60*15 + 1000*60*15
                            // the above rounds time down to a timestamp divisible by 15 minutes
                                )
                        ),
                        value
                ),
                /* your key serde */,
                /* your value serde */
        )
    .reduce(/*your adder*/, /*your subtractor*/, "store15m");

KTable<Windowed<Key>, Value> sixtyMinuteWindowed =

        fifteeenMinuteWindowed

    .groupBy( (windowedKey, value) ->
                new KeyValue<>(
                        new Windowed<>(
                                windowedKey.key(),
                                new Window<>(
                                        windowedKey.window().start() /1000/60/60 *1000*60*60,
                                        windowedKey.window().start() /1000/60/60 *1000*60*60 + 1000*60*60
                                        // the above rounds time down to a timestamp divisible by 60 minutes
                                )
                        ),
                        value
                ),
                /* your key serde */,
                /* your value serde */
        )

        .reduce(/*your adder*/, /*your subtractor*/, "store60m");
```

TODO: to mitigate infinite state store growth (until re-balance rebuilds it from the changelog) you can implement the Windowed key serde to store the timestamp(s) before the actual record key and periodically do a ranged query on each of the state stores to find and delete all data older than *x* (using punctuate() inside a Processor). TBC...